1. Exécution

À partir de la ligne de commande, on peut exécuter le programme appelé **script.py** de la façon suivante : **python script.py**.

Pour faire les exercices, vous pouvez modifier script.py dans n'importe quel éditeur de texte.

Dans un programme Python, les lignes qui commencent par un dièse (#) ne sont pas exécutées; ce sont des commentaires.

```
# Cette ligne est un commentaire
```

Exercice 1.1 : Ajouter un commentaire dans script.py et exécuter le programme.

2. Variables

Les variables sont utilisées pour stocker des valeurs dont on se servira par la suite. En Python, les noms de variables doivent commencer par une lettre. On affecte une valeur à une variable au moyen du symbole d'égalité (=).

```
a = 5
```

Il existe différents types de variables. Par exemple, une variable peut être un entier (*int*), un nombre réel (*float*) ou une chaîne de caractères (*str*).

```
a = 1
b = 1.5
c = "OLST"
```

Exercice 2.1 : Initialiser les variables *a*, *b* et *c* comme ci-dessus.

3. Print

La fonction **print** permet d'afficher des valeurs à l'écran.

```
print "Hello world!"
# Output : Hello world!
```

L'argument de la fonction **print** peut être une variable.

```
text = "Hello world!"
print text
# Output : Hello world!
```

N. B. Dans la version 3 de Python, l'argument de la fonction **print** doit être encadré de parenthèses.

```
print("Hello world!")
# Output : Hello world!
```

Exercice 3.1 : Faire afficher les variables *a*, *b* et *c* une après l'autre.

4. Arithmétique

On peut faire des opérations arithmétiques au moyen des opérateurs suivants :

- addition : +
- soustraction : -
- multiplication : *

- division : /
- exposant : **
- modulo : %

Le résultat d'une opération peut être stocké dans une variable.

```
resultat = 3 * 5
print resultat
# Output : 15
```

N. B. Dans la version 2 de Python, si on divise un entier par un entier, Python fait une division euclidienne (qui retourne un entier).

Exercice 4.1 : Diviser 3 par 2 et faire afficher le résultat. Ensuite, diviser 3.0 par 2 et faire afficher le résultat.

5. Comparaison

Les opérateurs de comparaison suivants permettent de comparer des valeurs :

- Est égal à : ==
- N'est pas égal à: !=
- Est plus petit que : <
- Est plus grand que : >
- Est plus petit ou égal à : <=
- Est plus grand ou égal à : >=

Ces comparaisons retournent une valeur de type *bool* (booléen). La valeur d'un booléen est soit True, soit False.

```
print 1==1
# Output : True
print 2<1
# Output : False
```

N. B. Ne pas confondre les opérateurs = (affectation) et == (égalité).

Exercice 5.1 : Au moyen de l'opérateur ==, faire une comparaison qui retourne False et stocker le résultat dans une variable.

Exercice 5.2 : Au moyen de l'opérateur <, faire une comparaison qui retourne True et stocker le résultat dans une variable.

6. Fonctions

Une fonction est un bout de code qui réalise une tâche particulière. Une fonction a un nom et peut prendre des *arguments*. Elle peut également retourner une valeur (en fait, elles retournent toujours quelque chose : en Python, une fonction qui ne retourne rien retourne en fait **None**, une constante qui représente l'absence de valeur).

Python fournit des fonctions prédéfinies, mais on peut également définir nos propres fonctions.

On peut vérifier le type d'une variable au moyen de la fonction **type()**, qui retourne le type de l'argument qu'on fournit entre parenthèses.

```
print type(a)
# Output : <type: 'int'>
```

Le résultat retourné par une fonction peut être stocké dans une variable.

```
t = type(a)
```

On peut convertir le type d'une variable au moyen de fonctions telles que **int()**, **str()** et **float()**.

```
s = str(3)
# s est la chaine "3"
s = int(s)
# s est l'entier 3
```

Exercice 6.1 : Faire afficher le type des variables *a*, *b* et *c*.

7. Fichiers

La fonction **open()** permet d'ouvrir un fichier. On s'en sert pour lire et écrire des fichiers. Cette fonction prend deux arguments : le nom d'un fichier et le mode de lecture. Pour lire un fichier, on utilise le mode "r" (pour read).

```
open("file.txt", "r")
```

La méthode **read()** permet de lire le contenu d'un fichier préalablement ouvert au moyen de la fonction **open()**.

```
file = open("file.txt", "r")
text = file.read()
```

Exercice 7.1 : Ouvrir le fichier data.txt et lire son contenu. Stocker le contenu du fichier dans une variable appelée *text*.

8. Listes

On utilise les crochets pour créer une structure de données qu'on appelle une *liste*. Les listes peuvent contenir n'importe quel type de variable, y compris d'autres listes. On peut créer une liste vide avec `[]` puis y ajouter des valeurs, ou l'initialiser en énumérant des valeurs, séparées par des virgules.

```
d = []
e = [1, 1.5, "OLST"]
```

On accède aux éléments d'une liste en écrivant le nom de la liste suivi d'un index entre crochets. En Python, l'index du premier élément d'une liste est 0.

```
f = e[0]
```

La méthode **append()** permet d'ajouter une valeur à la fin d'une liste.

```
list.append(value)
```

Autres méthodes qu'on peut appliquer aux listes : **index()**, **count()**, **sort()**, etc.

Autres structures de données : **dict**, **set**, **tuple**, etc.

Exercice 8.1 : Créer une liste contenant les valeurs "a","d" et "c" et la stocker dans une variable. Faire afficher le deuxième élément de cette liste. Ensuite, changer le deuxième élément de la liste par la valeur "b". Ensuite, ajouter la valeur "d" à la liste et faire afficher la liste au complet.

9. Manipulation de chaînes : Split et join

On peut appliquer à une chaîne la méthode **split()** pour la découper en sous-chaînes. Cette méthode

prend comme argument le délimiteur qu'on utilise pour découper la chaîne, et elle retourne la liste des sous-chaînes.

```
string = "1,2,3"
delimiter = ","
substrings = string.split(delimiter)
print substrings
# Output : ["1","2","3"]
```

La méthode **join()** concatène les chaînes contenues dans la liste qu'on lui fournit; on l'applique à la chaîne qu'on veut utiliser comme délimiteur (qui peut être la chaîne vide).

```
a_list = ["1","2","3"]
delimiter = ","
string = delimiter.join(a_list)
print string
# Output : "1,2,3"
```

Autres méthodes de manipulation de chaînes : **isalpha()**, **islower()**, **lower()**, **upper()**, **startswith()**, **find()**, **decode()**, **encode()**, etc.

Exercice 9.1 : Découper le contenu du fichier data.txt aux sauts de ligne (\n). Stocker le résultat dans une variable appelée *lignes*. Faire afficher cette variable.

Exercice 9.2 : Ajouter à la liste *lignes* la chaîne "azote,1".

10. Liste de listes

Une liste peut contenir d'autres listes. Par exemple, on peut représenter un tableau au moyen d'une liste de listes. Dans ce cas, chaque élément de la liste externe est lui-même une liste représentant une rangée. Pour accéder à une cellule du tableau, on doit fournir l'index d'une rangée (liste) parmi les éléments de la liste externe, suivi de l'index d'une des valeurs dans cette rangée (l'index de la colonne); les deux index sont encadrés de crochets.

```
table = [['a',1],['b',2],['c',3]]
row_1 = table[0]
print row_1
# Output : ['a',1]
cell_1_2 = table[0][1]
print cell_1_2
# Output : 1
```

11. Structures de contrôle

11.1 Boucle « for »

Exercice 11.1 : Faire afficher un par un tous les éléments dans la liste *lignes*. Commencer à l'index 0 et continuer jusqu'au dernier index.

On peut boucler sur les éléments d'une liste au moyen d'une structure de contrôle appelée *boucle for*.

```
for x in a_list:
    print x
```

Chaque valeur dans la liste sera affectée, une par une, à la variable x, et l'instruction à l'intérieur de la boucle sera exécutée pour chaque valeur de x. Ici, la variable dans laquelle on stocke, une par une, les valeurs dans la liste a le nom x, mais on peut la nommer comme on veut (tant que le nom commence par une lettre).

```
some_list = [1,2]
for i in some_list:
    print i
# Output : 1
# Output : 2
```

L'instruction à l'intérieur de la boucle doit être précédée d'une indentation (4 espaces).

On peut mettre plusieurs instructions à l'intérieur de la boucle for. Toutes les lignes qui ont la même indentation que la première instruction (ou un niveau plus élevé d'indentation) seront exécutées à chaque itération de la boucle. La première ligne dont le niveau d'indentation est inférieur à celui de la première instruction marque la fin du bloc d'instructions qui est executé à chaque itération de la boucle.

Exercice 11.2 : Faire afficher un par un tous les éléments dans la liste *lignes* au moyen d'une boucle.

Exercice 11.3 : Pour toutes les chaînes dans la liste *lignes*, découper la chaîne aux virgules, stocker le résultat dans une variable et faire afficher la première valeur dans cette liste.

Exercice 11.4 : Créer une liste vide appelée *table*. Pour toutes les chaînes dans la liste *lignes*, découper la chaîne aux virgules et ajouter le résultat (une liste) à la liste *table*. Le résultat sera une table à 2 colonnes, la première colonne contenant des mots, et la deuxième, des nombres (représentant la fréquence des mots).

Exercice 11.5 : Convertir toutes les valeurs (chaînes) dans la 2^{e} colonne de la table (liste de listes) appelée *table* en entiers.

Exercice 11.6 (facultatif) : Au moyen d'une boucle, compter le nombre de rangées dans *table* et faire afficher le résultat.

Exercice 11.7 (facultatif) : Au moyen d'une boucle, compter le nombre de colonnes dans *table* et faire afficher le résultat.

Exercice 11.8 (facultatif) : Créer une liste appelée *nombres* contenant les entiers 1 à 5. Au moyen d'une boucle, calculer la somme de ces nombres.

11.2 Instruction conditionnelle (*if*)

On peut vérifier si une condition est vraie avant d'exécuter une instruction au moyen d'une instruction conditionnelle (un *if*).

```
if condition:
    do something
```

La condition peut être le résultat d'une comparaison; par exemple, on peut vérifier si une variable a une valeur particulière.

```
if 1 == 2:
    print "Success"
# Output :
if 1 < 2:
    print "Success"
# Output : Success
a = 1
if a == 1:
    print "Success"
# Output : Success
```

N. B. Les valeurs suivantes sont équivalentes à False : None, 0 (ainsi que 0.0 et 0j), la chaîne vide (""), la liste vide ([]), le tuple vide ((), le dictionnaire vide ({}) et toute classe dont la méthode __bool__ retourne False ou dont la méthode __len__ retourne 0. Toute autre valeur est équivalente à True.

On peut aussi préciser quoi faire si la condition est fausse au moyen de **else**.

```
a = 3
if a > 5:
    print "Big"
else:
    print "Small"
# Output : Small
```

On peut utiliser des instructions conditionnelles à l'intérieur d'une boucle for, ou vice-versa. Dans ces cas, il faut utiliser plus d'un niveau d'indentation.

```
some_list = [1,2,3]
for value in some_list:
    if value > 2:
        print value
# Output : 3
```

On peut trouver la valeur la plus élevée dans une liste au moyen d'une boucle et d'une instruction conditionnelle.

```
max_value = 0
some_list = [1,2,3]
for value in some_list:
    if value > max_value:
        max_value = value
print max_value
# Output = 3
```

Exercice 11.2 : Faire afficher tous les mots dans data.txt dont la fréquence est exactement 6.

Exercice 11.3 : Faire afficher tous les mots dans data.txt dont la fréquence est supérieure à 10.

Exercice 11.4 : Trouver le mot dont la fréquence est la plus élevée. Faire afficher le mot et sa fréquence.

12. Le mode interactif

On peut écrire et exécuter du code de manière interactive au moyen du mode interactif de python. On le lance en utilisant la commande **python** (sans argument).

13. Référence

Ce tutoriel est basé sur le cours sur Python pour débutants offert par DataQuest (https://www.dataquest.io/).

14. Ressources

- Cours en ligne : https://www.dataquest.io/, https://www.codecademy.com/, https://www.coursera.org/, https://www.udacity.com/, https://www.edx.org/, etc.
- Tutoriel officiel : https://docs.python.org/2/tutorial/index.html.
- Apprendre par la résolution de problèmes : http://www.pythonchallenge.com/, https://projecteuler.net/, etc.

- Livres : A Byte of Python (C. H. Swaroop), Learning Python (M. Lutz), Learn Python the Hard Way (Z. A. Shaw), Natural Language Processing with Python (S. Bird, E. Klein & E. Loper), etc.
- Forums : http://stackoverflow.com/.